

Deep Radial Kernel Networks: Approximating Radially Symmetric Functions with Deep Networks

Brendan McCane and Lech Szymanski

Department of Computer Science
University of Otago
Dunedin, New Zealand.

March 13, 2017

Abstract

We prove that a particular deep network architecture is more efficient at approximating radially symmetric functions than the best known 2 or 3 layer networks. We use this architecture to approximate Gaussian kernel SVMs, and subsequently improve upon them with further training. The architecture and initial weights of the Deep Radial Kernel Network are completely specified by the SVM and therefore sidesteps the problem of empirically choosing an appropriate deep network architecture.

1 Introduction

Deep networks have been stunningly successful in many machine learning domains since the area was reinvigorated by the work of Krizhevsky et al. [2012]. Despite their success, neural networks in general, have two major limitations:

1. relatively little is known about them theoretically, although this is changing. In particular, for which problems are deep networks more effective than shallow learners; and
2. deciding on a particular architecture is still an empirical question.

Compare using a deep network for a particular problem with a support vector machine (SVM). The choices for a deep network include: number of layers, number of neurons per layer, activation function, gradient descent algorithm, gradient descent algorithm parameters (e.g. learning rate), which learning tricks to use (dropout, batch normalisation etc), weight initialisation parameters, etc. Getting any one of these wrong can cause the network learning to fail. For an SVM, on the other hand, one

needs to choose the kernel function and the kernel function parameters (usually few) and the slack variable parameter C . Similarly, other machine learning methods such as decision trees, boosting, random forests are much easier to use, albeit often with worse performance than the best neural network model.

This paper goes some way to addressing both of these limitations and offers a theoretical result and an applied result. The main theoretical result is a new upper bound for the number of neurons required in a deep network to approximate a radially symmetric function. The main applied result uses the theoretical result to construct a deep network approximation of an SVM that can be further trained using back propagation and which often results in improved performance.

2 Related Work

It is difficult to deny the empirical success of deep networks. It is still the case that relatively little is known theoretically about their fundamental abilities, although this has been changing over the last few years. Nevertheless, most work consists of existence proofs that do not lead directly to new methods for building or training networks. For example, ReLU networks with n_0 inputs, L hidden layers of width $n \geq n_0$ can compute functions that have $\Omega((n/n_0)^{(L-1)n_0} n^{n_0})$ linear regions compared to $\sum_{j=0}^{n_0} \binom{n}{j}$ for a shallow network [Montufar et al., 2014]. More generally, Telgarsky [2016] proved for semi-algebraic neurons (including ReLU, sigmoid etc), that networks exist with $\Theta(k^3)$ layers and $\Theta(k^3)$ nodes that require $\Omega(2^k)$ nodes to approximate with a network of $O(k)$ layers. Delalleau and Bengio [2011] show that deep sum-product networks exist for which a shallow network would require exponentially more neurons to simulate. For convolutional arithmetic circuits (similar to sum-product networks), Cohen et al. [2016], in an important result, show that “besides a negligible (zero measure) set, all functions that can be realized by a deep network of polynomial size, require exponential size in order to be realized, or even approximated, by a shallow network.”

The above works, except for Cohen et al. [2016] focus on approximating deep networks with shallow networks, but do not indicate what problems are best attacked with deep networks. For manifolds, Basri and Jacobs [2016] show how deep networks can efficiently represent low-dimensional manifolds and that these networks are almost optimal, but they do not discuss limitations of shallow networks on the same problem. Somewhat similarly, Shaham et al. [2016] show that depth-4 networks can approximate a function on a manifold where the number of neurons depends on the complexity of the function and the dimensionality of the manifold and only weakly on the embedding dimension. Again, they do not discuss the limitations of shallow networks for this problem. Importantly, both of these results are constructive and allow one to actually build the network.

Szymanski and McCane [2014] show that deep networks can approximate periodic functions of period P over $\{0, 1\}^N$ with $O(\log_2 N - \log_2 P)$ parameters versus $O(P \log_2 N)$ for shallow. Eldan and Shamir [2016] show that 3-layer networks exist such that the network can approximate a radially symmetric function with $O(d^{19/4})$ neurons, that a 2-layer network requires at least $O(e^d)$ neurons.

Therefore evidence is building that deep networks are more powerful than their

shallow counterparts in terms of the number of parameters or neurons required. Nevertheless, we shouldn't stop there because there is relatively little work linking this theory with practical applications of the same. In this work we directly extend the work of Szymanski and McCane [2014] and Eldan and Shamir [2016]. The latter work [Eldan and Shamir, 2016] is extended to deeper networks for approximating radially symmetric functions that require fewer parameters than their construction. The former Szymanski and McCane [2014] is extended by generalising their notion of folding transformations to work in multiple dimensions and more simply with ReLU networks. The proofs are constructive and allow us to build networks for approximating radially symmetric functions. These networks are used to approximate Gaussian kernel SVMs and the results show how we can further train these approximations to do better than the original SVM in many cases.

Our applied work is similar in spirit, but quite different in detail to other methods trying to combine deep learning and kernel learning. For example, Wilson et al. [2016] use a deep network as the kernel in a Gaussian processes framework, but the choice of the deep network architecture remains with the practitioner. Our deep radial kernel could be used as the input kernel in that framework. We also note that multiple kernel learning [Lanckriet et al., 2004] and its descendants take a very different approach to the one we take here — we make no attempt to learn a kernel matrix, nor to ensure positive semi-definiteness of the resulting network.

We start in Section 3 with the main theoretical results, then proceed to approximating Gaussian kernel SVMs in Section 4.

3 Theory

3.1 Context and Notation

A radially symmetric function is a function whose value is dependent on the norm of the input only. We are interested in L -Lipschitz functions f , $|f(x) - f(y)| \leq L|x - y|$, as this covers many functions common in classification tasks. The number of dimensions of the input is d , and we assume that f is constant outside a radius R . This is a similar context to that used by Eldan and Shamir [2016]. Further, we restrict ourselves to ReLU networks only, which is more restrictive than Eldan and Shamir [2016], but allows us to explicitly construct the networks of interest.

Most proofs are only sketched in the main body of the paper. Detailed proofs are provided in the supplementary material.

3.2 3 Layer Networks

We start by stating a modified form of Lemma 18 from Eldan and Shamir [2016]:

Lemma 1. *Modified form of Lemma 18 from Eldan and Shamir [2016]:*

Let $\sigma(z) = \max(0, z)$. Let f be an L -Lipschitz function supported on $[0, R]$. Then for any $\delta > 0$, there exists a function g expressible by a 3-layer network of width at most

$\frac{6d^2R^2+3RL}{\delta}$, such that

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\|\mathbf{x}\|)| < \delta + L\sqrt{\delta}.$$

The proof follows the basic plan of Eldan and Shamir [2016] where the first layer is the input layer, the second layer approximates x_i^2 for each dimension i , and the third layer computes $\sum_i x_i^2$ and approximates f .

Since several sections of the second layer are doing the same thing (computing the square of their input), a weight-sharing corollary follows immediately where only one copy of the square approximation is needed.

Corollary 1 (3 Layer Weight Sharing). *Let $\sigma(z) = \max(0, z)$. Let f be an L -Lipschitz function supported on $[0, R]$. Then for any $\delta > 0$, there exists a function g expressible by a 3-layer weight-sharing network with at most $\frac{6dR^2+3RL}{\delta}$ weights, such that*

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\|\mathbf{x}\|)| < \delta + L\sqrt{\delta}.$$

3.3 Deep Folding Networks

In this section we show how folding transformations can be used to create a much deeper network with the same error, but many fewer weights than needed in Lemma 1. A folding transformation is one in which half of a space is reflected about a hyper-plane, and the other half remains unchanged. Figure 1 demonstrates how a sequence of folding transformations can transform a circle in 2D to a small sector. After enough folds, we can discard the almost zero coordinates to approximate the norm. We will use this general idea to prove the following theorem:

Theorem 1. *Let $\mathbf{x} \in \mathbb{R}^d$, and $\sigma(z) = \max(0, z)$. Let f be an L -Lipschitz function supported on $[0, R]$. Fix $L, \delta, R > 0$. There exists a function g expressible by a $O(d \log_2(d) + \log_2(d) \log_2(\frac{R}{\sqrt{\delta}}))$ layer network where the number of weights, and number of neurons, $N_w, N_n = O(d^2 + d \log_2(\frac{R}{\sqrt{\delta}}) + \frac{3RL}{\delta})$, such that:*

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\|\mathbf{x}\|)| < L\sqrt{\delta} + \delta$$

The approach taken here is a constructive one and specifies the architecture of the network needed to approximate f . In fact, all of the weights except those in the last layer are specified. The approach is somewhat different to that used to prove Lemma 1. We build a sequence of layers to directly approximate $\|\mathbf{x}\|$ and then approximate f in the last layer. To build our layers, we need a few helper lemmas.

Lemma 2 (2D fold). *There exists a function $g : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, expressible by a ReLU network with 4 ReLU units and 2 sum units that can compute a folding transformation about a line through the origin, represented by the unit direction vector $\mathbf{l} = (l_x, l_y)^T$.*

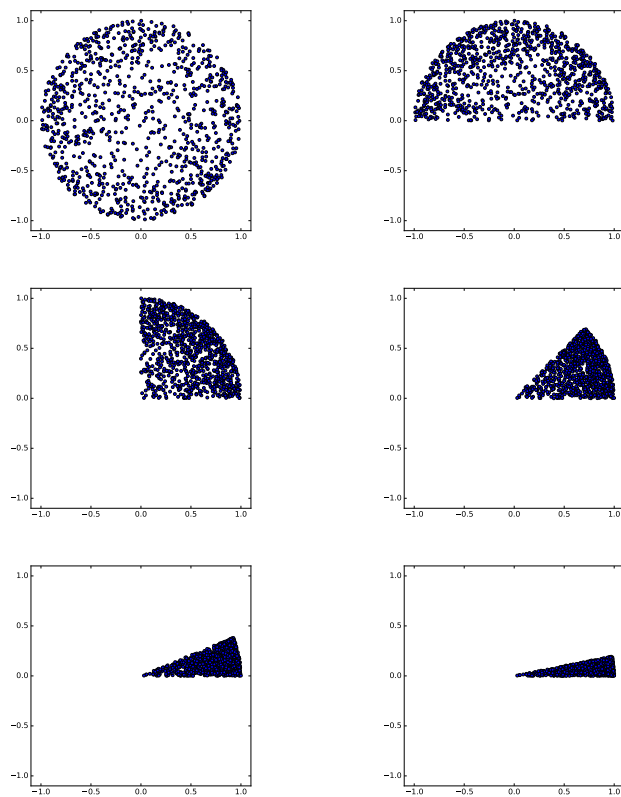


Figure 1: Series of folding transformations for 2D.

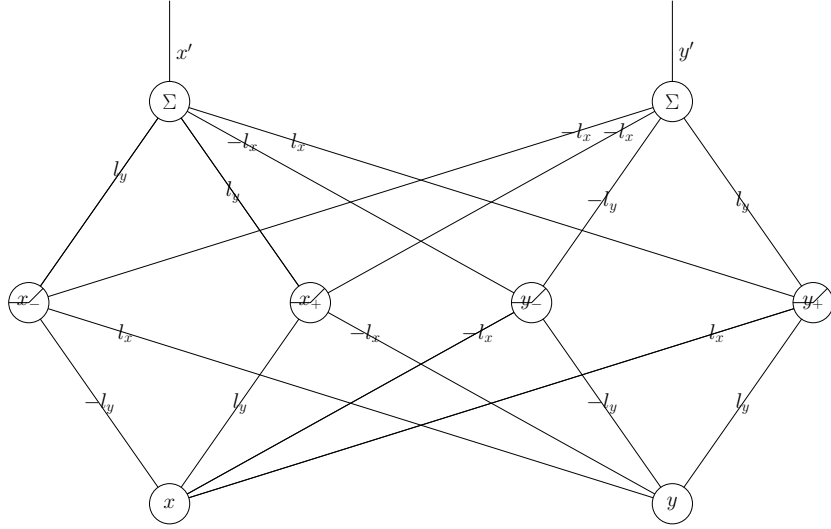


Figure 2: A network to produce a 2D fold.

The function g is of the form:

$$g(\mathbf{x}) = \begin{cases} \mathbf{x} & \mathbf{l} \cdot \mathbf{x}^\perp > 0 \\ \begin{bmatrix} l_x^2 - l_y^2 & 2l_x l_y \\ 2l_x l_y & l_y^2 - l_x^2 \end{bmatrix} \mathbf{x} & \text{otherwise.} \end{cases}$$

The requisite ReLU network is shown in Figure 2. Only one of the nodes labeled x_- (y_-) and x_+ (y_+) are active at any one time. Therefore there are four possible cases depending on which two nodes are active. Note that x_- is active when $\mathbf{l} \cdot \mathbf{x}^\perp < 0$ and x_+ is active when $\mathbf{l} \cdot \mathbf{x}^\perp > 0$. To approximate the 2D norm, we simply stack layers of the type shown in Figure 2 with suitable choice of l_x, l_y at each layer. Note that the summation nodes aren't required since they can be incorporated into the summations and weights of the next ReLU layer. These 2D folds can be used to estimate the norm of a vector as per the following lemma.

Lemma 3 (Approximate $\|\mathbf{x}\|, \mathbf{x} \in \mathbb{R}^2, \|\mathbf{x}\| < R$). *There exists a function g , expressible by a ReLU network with no more than $\log_2(R/\delta)$ layers and 4 nodes per layer such that:*

$$\sup_{\mathbf{x} \in \mathbb{R}^2, \|\mathbf{x}\| \leq R} |g(\mathbf{x}) - \|\mathbf{x}\|| \leq \delta$$

Proof. The proof is short and simple. After f layers, each data point will be within an

angle of $\frac{\pi}{2^f}$ of the x -axis. Simple geometry and appropriate approximations leads to:

$$\begin{aligned}
\delta &= \|\mathbf{x}\| - \|\mathbf{x}\| \cos\left(\frac{\pi}{2^f}\right) \\
&\leq R \left(1 - \cos\left(\frac{\pi}{2^f}\right)\right) \\
&\leq R \left(2 \sin\left(\frac{\pi}{2^{f+1}}\right)\right) \\
&\leq R \left(\frac{\pi}{2^f}\right) \\
f &\leq \log_2 \left(R \frac{\pi}{\delta}\right)
\end{aligned}$$

□

The following lemma generalises this construction to folds in d dimensions.

Lemma 4 (Approximate $\|\mathbf{x}\|$, $\mathbf{x} \in \mathbb{R}^d$). *There exists a function g , expressible by a ReLU network with:*

$$\begin{aligned}
N_l &\leq \log_2(d) \\
&\log_2 \left(\frac{R\pi}{\delta} \left[(2^{\lfloor \frac{d+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{d}{2} \rfloor} - 1) \right] \right) \\
N_n &\leq 4(d-1) \\
&\log_2 \left(\frac{R\pi}{\delta} \left[(2^{\lfloor \frac{d+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{d}{2} \rfloor} - 1) \right] \right) \\
N_w &\leq 8(d-1) \\
&\log_2 \left(\frac{R\pi}{\delta} \left[(2^{\lfloor \frac{d+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{d}{2} \rfloor} - 1) \right] \right)
\end{aligned}$$

such that:

$$\sup_{\mathbf{x} \in \mathbb{R}^d, \|\mathbf{x}\| \leq R} |g(\mathbf{x}) - \|\mathbf{x}\|| \leq \delta$$

We note that a fold in a 2D plane in \mathbb{R}^d will leave all coordinates perpendicular to the plane unchanged. We can therefore apply the approximation of Lemma 3 to pairs of input coordinates to produce $d/2$ new coordinates. Then apply the same reduction to produce $d/4$ coordinates and continue on this way until there is only one coordinate left. In effect, we are calculating the norm via the following scheme:

$$\begin{aligned}
&\sqrt{x_1^2 + x_2^2 + x_3^2 + \dots + x_d^2} = \\
&\sqrt{\sqrt{\sqrt{x_1^2 + x_2^2}^2 + \sqrt{x_3^2 + x_4^2}^2}^2 \dots \sqrt{\dots + \sqrt{x_{n-1}^2 + x_{n-2}^2}^2}^2}
\end{aligned}$$

Figure 3 shows the resulting network. The proof is by induction and is rather long so is not produced here but appears in full in the supplementary material.

At this point we make use of Lemma 19 from Eldan and Shamir [2016] which we reproduce here:

Lemma 5 (Lemma 19 from Eldan and Shamir [2016]). *Let $\sigma(z) = \max(0, z)$ be the ReLU activation function, and fix $L, \delta, R > 0$. Let $f : \mathbb{R} \rightarrow \mathbb{R}$ which is constant outside an interval $[-R, R]$. There exist scalars $a, \{\alpha_i, \beta_i\}_{i=1}^w$, where $w \leq 3\frac{RL}{\delta}$, such that the function:*

$$h(x) = a + \sum_{i=1}^w \alpha_i \sigma(x - \beta_i) \quad (1)$$

is L -Lipschitz and satisfies:

$$\sup_{x \in \mathbb{R}} |h(x) - f(x)| \leq \delta. \quad (2)$$

Moreover, one has $|\alpha_i| \leq 2L$ and $w \leq 3\frac{RL}{\delta}$.

We can now prove the main theorem (some steps left out for brevity):

Theorem 1. *Let $\mathbf{x} \in \mathbb{R}^d$, and $\sigma(z) = \max(0, z)$. Let f be an L -Lipschitz function supported on $[0, R]$. Fix $L, \delta, R > 0$. There exists a function g expressible by a $O(d \log_2(d) + \log_2(d) \log_2\left(\frac{R}{\sqrt{\delta}}\right))$ layer network where the number of weights, and number of neurons, $N_w, N_n = O(d^2 + d \log_2\left(\frac{R}{\sqrt{\delta}}\right) + \frac{3RL}{\delta})$, such that:*

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\|\mathbf{x}\|)| < L\sqrt{\delta} + \delta$$

Proof. From Lemma 4 we can approximate $\|\mathbf{x}\|$ to within $\sqrt{\delta}$ and using Lemma 5:

$$f(\|\mathbf{x}\|) + L\sqrt{\delta} - \delta \leq g(\|\mathbf{x}\| + \sqrt{\delta}) \leq f(\|\mathbf{x}\|) + L\sqrt{\delta} + \delta$$

and

$$f(\|\mathbf{x}\|) - L\sqrt{\delta} - \delta \leq g(\|\mathbf{x}\| + \sqrt{\delta}) \leq f(\|\mathbf{x}\|) - L\sqrt{\delta} + \delta$$

therefore:

$$f(\|\mathbf{x}\|) - L\sqrt{\delta} - \delta \leq g(\|\mathbf{x}\| + \sqrt{\delta}) \leq f(\|\mathbf{x}\|) + L\sqrt{\delta} + \delta$$

The number of weights and neurons required by Lemma 5 is $3\frac{RL}{\delta}$. The number of weights and neurons required to estimate $\|\mathbf{x}\|$ is given by Lemma 4 (substituting $\sqrt{\delta}$ for δ). Stack the network from Lemma 5 (1 layer) onto the end of the network from Lemma 4 ($O(d \log_2(d) + \log_2(d) \log_2(R/\sqrt{\delta}))$ layers), thus requiring a total number of neurons no more than:

$$\begin{aligned} N_n &\leq \left[4(d-1) \right. \\ &\quad \left. \log_2 \left(\frac{R\pi}{\sqrt{\delta}} \left[(2^{\lfloor \frac{d-1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{d}{2} \rfloor} - 1) \right] \right) \right] \\ &\quad + \frac{3RL}{\delta} \\ N_n &= O \left(d^2 + d \log_2 \left(\frac{R}{\sqrt{\delta}} \right) + \frac{3RL}{\delta} \right) \end{aligned}$$

and a total number of weights no more than:

$$\begin{aligned}
N_w &\leq \left[8(d-1) \right. \\
&\quad \left. \log_2 \left(\frac{R\pi}{\sqrt{\delta}} \left[(2^{\lfloor \frac{d-1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{d}{2} \rfloor} - 1) \right] \right) \right] \\
&\quad + \frac{3RL}{\delta} \\
N_w &= O \left(d^2 + d \log_2 \left(\frac{R}{\sqrt{\delta}} \right) + \frac{3RL}{\delta} \right)
\end{aligned}$$

□

Again, there is an obvious weight-sharing corollary:

Corollary 2 (Deep weight sharing network). *Let $\mathbf{x} \in \mathbb{R}^d$, and $\sigma(z) = \max(0, z)$. Let f be an L -Lipschitz function supported on $[0, R]$. Fix $L, \delta, R > 0$. There exists a function g expressible by a network where the number of weights is at most $N_w = O \left(d + \log_2 \left(\frac{R}{\sqrt{\delta}} \right) + \frac{3RL}{\delta} \right)$, such that:*

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\|\mathbf{x}\|)| < L\sqrt{\delta} + \delta$$

Comparing Theorem 1 to Lemma 1, both are of order d^2 , however the folding network version is more efficient in terms of R and δ . This means the deeper network can be much more efficient when either d or R is large, or δ is small.

4 Deep Radial Kernel Network (DRKN)

We have shown constructively that a deep network using folding transformations can more efficiently represent finite extent radially symmetric functions than a corresponding 3-layer network. This construction can therefore be used to approximate any system that makes use of radial functions, and could be particularly useful for approximating radially symmetric Gaussians. There are advantages and disadvantages to doing so. The main advantage is that it allows the power and flexibility of deep neural networks to be applied in a systematic way with the architecture specified by the problem at hand. The disadvantage is that it can be more computationally costly to evaluate the radial functions via a deep network compared to directly using the function itself. However, this cost is offset by the extra flexibility afforded by the deep structure. After initialization, the network can be further trained, allowing it to adapt more towards the data and away from the radial function approximation. In this section we demonstrate this idea on support vector machines with Gaussian kernels and show empirically that such a construction tends to perform better than the corresponding SVM but does not appear to suffer greatly from overtraining.

The first step in creating a DRKN is to train a support vector machine. The method will work for any SVM (or support vector regression) that uses a radially symmetric

kernel. The most popular is the Gaussian kernel and that is what we use here. In this case, for multi-class problems, we use one-vs-rest SVMs. The decision function of the SVM is:

$$f(\mathbf{X}) = \arg \max_{1 \leq c \leq N_C} \sum_{i=1}^{N_{C,V}} \alpha_{c,i} K(V_{c,i}, X), \quad (3)$$

where N_C is the number of classes, $N_{C,V}$ is the number of support vectors for class C , $\alpha_{c,i}$ is the coefficient for support vector $V_{c,i}$, and $K(\cdot, \cdot)$ is the kernel function. Note that $\alpha_{c,i}$ can be positive or negative as it incorporates the class label of the relevant binary classification problem (1 for the class of interest, and -1 for all other classes).

An SVM with a Gaussian kernel has two parameters: σ which specifies the width of the kernel; and C , the trade-off between misclassification and decision surface smoothness. These parameters need to be estimated or specified to train an SVM. We make use of the Python scikit learn package [Pedregosa et al., 2011] and a randomised search over an exponential distribution to estimate the optimal parameters for the SVM.

There are several ways to convert Equation 3 into a deep network using the techniques of Section 3.3, but for this paper we use the most direct method. The majority of the network relates to approximating the kernel. This kernel is weight shared across all support vectors as in the support vector machine (one alternative is to have a different kernel for each support vector). The kernel used for training the SVM is a Gaussian kernel, however Section 3.3 requires a kernel of finite support. For the DRKN we approximate the Gaussian kernel using the polynomial method of Fornefett et al. [2001][$Q_{3,1}$] first, then approximate the polynomial using the method of Section 3.3. The network implements the following decision function:

$$f'(\mathbf{X}) = \arg \max_{1 \leq c \leq N_C} \left[\frac{1}{2} + \frac{1}{2} \tanh \left(\sum_{i=1}^{N_{C,V}} \alpha_{c,i} F_n(V_{c,i} - X) \right) \right], \quad (4)$$

where F_n is the fold network approximation. We use the cross-entropy softmax loss function and optimise over all weights in the fold network, the support vector centres, and the support vector weights. A conceptual diagram of part of the network for one class is given in Figure 4.¹ The cross-entropy objective function is used with stochastic gradient descent. The number of samples in each mini-batch varies depending on the problem, and ranges from 10 to 100.

4.1 Datasets

Several standard datasets have been used to test the algorithm and these are listed in Table 1. Most datasets were sourced from the UCI machine learning repository, with the covtype dataset coming via the Python module sklearn. If the dataset was already split into train and test sets, then our testing made use of these sets. If not, then the training set was split 70/30 into train and test sets except for the covtype dataset. In that case, there were too many samples for effective training of an SVM, so 100000

¹Code for approximating an SVM and training a DRKN can be downloaded from <https://bitbucket.org/mccane/deep-radial-kernel-network>.

Name	Source	Dims	Training #	Test #
svmguide1	UCI	4	3090	400
whitewine	UCI	11	3428	1470
redwine	UCI	8	1112	480
breastcancer	UCI	10	490	210
sat	UCI	37	4435	2000
sensorless drive	UCI	48	46808	11702
segmentation	UCI	20	1617	694
covtype	sklearn	54	70000	30000

Table 1: Datasets used to test the algorithm and compare with SVM.

data points were randomly sampled from the set, and these were split 70/30 into train and test sets. In all cases, the SVM and deep network were trained and tested on the same data.

4.2 Results

Figure 5 shows the results for the 8 example problems with train and test error shown as the number of epochs increases. For reference we include the SVM test error and training and test error for a radial basis function (RBF) network that was initialised with the same kernel and support vectors as the SVM - essentially replacing the fold network and the function approximate in Figure 4 with a Gaussian RBF neuron. For the RBF network the Gaussian parameter, the support vectors and all the network weights are trainable. The RBF network was included to test whether moving the support vectors and/or adjusting the width of the Gaussians were the factors producing improvement.

Note that there is a training anomaly in the covtype results for the DRKN. We think this is due the optimisation algorithm taking a misstep but quickly recovering. In any case, it has little effect on the long term results.

5 Discussion

We have derived a new upper bound for deep networks approximating radially symmetric functions and have shown that deeper networks are more efficient than the 3-layer network of Eldan and Shamir [2016]. The central concept in this construction is a space fold — halving the volume of space that each subsequent layer needs to handle. We hypothesise that to take full advantage of deep networks we need to apply operations that work on multiple areas of the input space simultaneously, analogous to taking advantage of the multiple linear regions noted by Montufar et al. [2014]. Folding transformations are one way to ensure that any operation applied in a later layer is simultaneously applied to many regions in the input layer. We believe there are many other possible transformations, but the reflections used here might be considered fundamental in a sense due to the Cartan-Dieudonné-Scherk Theorem which states that all orthogonal transformations can be decomposed into a sequence of reflections. We are yet to fully investigate the consequences of this theorem.

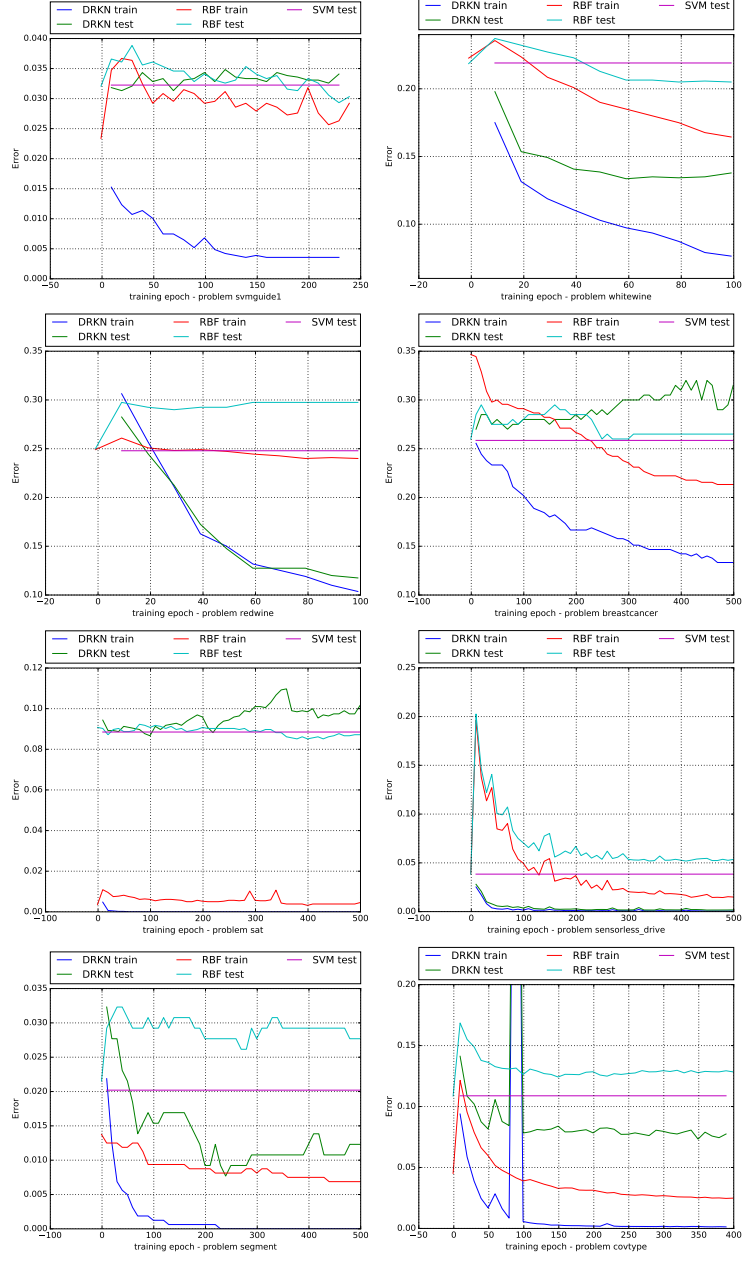


Figure 5: Errors and training epoch for each of the example problems.

Radial basis function (RBF) networks are universal approximators and therefore we can use deep radial approximations to approximate any function by decomposing the function into a sum of RBFs first (such as mixture of Gaussians). However, the efficiency of the method will decrease as the number of RBFs required increases. It remains an open question at which point it becomes more efficient to directly approximate the function with a shallow network.

We have used our theoretical construction to build a deep radially symmetric function approximator and employed it to approximate Gaussian kernel SVMs. Of the 8 problems tested, the method performs much better on 3, moderately better on two, and similarly on 3. In contrast, an actual RBF network approximation has performed no better than the initial SVM. It should be pointed out that there is no real loss, other than computation, to trying the DRKN on any particular problem. It requires no parameter tuning other than the choice of optimisation method and learning parameters, and if it performs better it can be adopted, but if it does not, one can always fall back to the original SVM. The flexibility of the DRKN comes at a cost however, as the size of the network can be quite large, and hence training can be slow. However, the method is equally applicable to more scalable approximate SVM algorithms such as the Core Vector Machine [Tsang et al., 2005] or SimpleSVM [Vishwanathan and Murty, 2002], and future work will involve testing the method on these approximate algorithms.

References

- Ronen Basri and David Jacobs. Efficient representation of low-dimensional manifolds using deep networks. *arXiv preprint arXiv:1602.04723*, 2016.
- Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: a tensor analysis. *JMLR: Workshop and Conference Proceedings*, 49:1–31, 2016.
- Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674, 2011.
- Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. *JMLR: Workshop and Conference Proceedings*, 49:134, 2016.
- Mike Fornefett, Karl Rohr, and H Siegfried Stiehl. Radial basis functions with compact support for elastic registration of medical images. *Image and vision computing*, 19(1):87–96, 2001.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- Gert RG Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine learning research*, 5(Jan):27–72, 2004.

- Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2924–2932, 2014.
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- Uri Shaham, Alexander Cloninger, and Ronald R Coifman. Provable approximation properties for deep neural networks. *Applied and Computational Harmonic Analysis*, 2016.
- Lech Szymanski and Brendan McCane. Deep networks are effective encoders of periodicity. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(10): 1816–1827, 2014.
- Matus Telgarsky. Benefits of depth in neural networks. *JMLR: Workshop and Conference Proceedings*, 49:1–23, 2016.
- Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6 (Apr):363–392, 2005.
- SVM Vishwanathan and M Narasimha Murty. Ssvm: a simple svm algorithm. In *Neural Networks, 2002. IJCNN’02. Proceedings of the 2002 International Joint Conference on*, volume 3, pages 2393–2398. IEEE, 2002.
- Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 370–378, 2016.

Appendices

A Some Preliminaries

First a definition of L -Lipschitz. A function is L -Lipschitz if:

$$|f(x) - f(y)| \leq L|x - y| \quad (5)$$

We also need to make use of the following Lemma from Eldan and Shamir [2016] which we state without proof:

Lemma 5 (Lemma 19 from Eldan and Shamir [2016]). *Let $\sigma(z) = \max(0, z)$ be the ReLU activation function, and fix $L, \delta, R > 0$. Let $f : \mathbb{R} \rightarrow \mathbb{R}$ which is constant outside an interval $[-R, R]$. There exist scalars $a, \{\alpha_i, \beta_i\}_{i=1}^w$, where $w \leq 3\frac{RL}{\delta}$, such that the function:*

$$h(x) = a + \sum_{i=1}^w \alpha_i \sigma(x - \beta_i) \quad (1)$$

is L -Lipschitz and satisfies:

$$\sup_{x \in \mathbb{R}} |h(x) - f(x)| \leq \delta. \quad (2)$$

Moreover, one has $|\alpha_i| \leq 2L$ and $w \leq 3\frac{RL}{\delta}$.

B 3 Layer Network

We also need the following lemma which is modified from Lemma 18 of [Eldan and Shamir, 2016]. Since it is a modified version, we give a proof.

Lemma 1. *Modified form of Lemma 18 from Eldan and Shamir [2016]:*

Let $\sigma(z) = \max(0, z)$. Let f be an L -Lipschitz function supported on $[0, R]$. Then for any $\delta > 0$, there exists a function g expressible by a 3-layer network of width at most $\frac{6d^2 R^2 + 3RL}{\delta}$, such that

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\|\mathbf{x}\|)| < \delta + L\sqrt{\delta}.$$

Proof. The proof consists of constructing a 3-layer network with the first layer being the input. The second layer approximates x_i^2 , then the third layer computes $\sum_i x_i^2$ and approximates f .

Approximate x_i^2 :

Define the $2R$ -Lipschitz function:

$$l(x) = \min\{x^2, R^2\}, \quad (6)$$

Using Lemma 5, create the function $\bar{l}(x)$ having the form $a + \sum_{i=1}^{w_1} \alpha_i \sigma(x - \beta_i)$ so that

$$\sup_{x \in \mathbb{R}} |\bar{l}(x) - l(x)| \leq \frac{\delta}{d} \quad (7)$$

where $w_1 \leq \frac{6dR^2}{\delta}$.

Approximate $\sum_i x_i^2$:

Define the function $\ell : \mathbb{R}^d \rightarrow \mathbb{R}$

$$\ell(\mathbf{x}) = \sum_{i=1}^d l(x_i) = \sum_{i=1}^d \min\{x_i^2, R^2\} \quad (8)$$

$\ell(\mathbf{x})$ is $2dR$ -Lipschitz. Consequently, define the function

$$\bar{\ell}(\mathbf{x}) = \sum_{i=1}^d \bar{l}(x_i) = \sum_{i=1}^d \left[a_i + \sum_{j=1}^{w_1} \alpha_{ij} \sigma(x_i - \beta_{ij}) \right] \quad (9)$$

Note that $\bar{\ell}(\mathbf{x})$ is $2dR$ -Lipschitz. At this point we have $w_{12} = dw_1 \leq \frac{6d^2R^2}{\delta}$ weights in the first two layers of the network and:

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |\bar{\ell}(\mathbf{x}) - \ell(\mathbf{x})| \leq \delta. \quad (10)$$

Approximate $f(\|\mathbf{x}\|)$:

The input to the final layer is $\bar{\ell}(\mathbf{x})$ which is an approximation of $\ell(\mathbf{x}) = \|\mathbf{x}\|^2$. The error associated with approximating $f(\sqrt{\ell(\mathbf{x})})$ is:

$$|f(\sqrt{\bar{\ell}(\mathbf{x})}) - f(\sqrt{\ell(\mathbf{x})})| \leq L|\sqrt{\bar{\ell}(\mathbf{x})} - \sqrt{\ell(\mathbf{x})}| \quad (11)$$

$$\leq L|\sqrt{\bar{\ell}(\mathbf{x})} \pm \sqrt{\delta} - \sqrt{\ell(\mathbf{x})}| \quad (12)$$

$$\leq L|\sqrt{\delta}| \quad (13)$$

Since f is L -Lipschitz, $\ell(\mathbf{x}) - \delta \leq \bar{\ell}(\mathbf{x}) \leq \ell(\mathbf{x}) + \delta$ and $\sqrt{\bar{\ell}(\mathbf{x})} \pm \sqrt{\delta} \leq \sqrt{\ell(\mathbf{x})} \pm \sqrt{\delta}$. Now we are able to approximate $f(\sqrt{\bar{\ell}(\mathbf{x})})$ using Lemma 5 with a function of the form:

$$g(\mathbf{x}) = a + \sum_{k=1}^{w_3} \alpha_k \sigma \left(\sum_{i=1}^d \left[a_i + \sum_{j=1}^{w_1} \alpha_{ij} \sigma(x_i - \beta_{ij}) \right] - \beta_k \right) \quad (14)$$

From Lemma 5:

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\sqrt{\bar{\ell}(\mathbf{x})})| \leq \delta \quad (15)$$

with $w_3 \leq \frac{3RL}{\delta}$. So either (taking the worst case error):

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\sqrt{\ell(\mathbf{x})})| = \sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\sqrt{\bar{\ell}(\mathbf{x})}) - L\sqrt{\delta}| \quad (16)$$

$$= \sup_{\mathbf{x} \in \mathbb{R}^d} ||g(\mathbf{x}) - f(\sqrt{\bar{\ell}(\mathbf{x})})| - L\sqrt{\delta}| \quad (17)$$

$$\leq |\delta - L\sqrt{\delta}| \quad (18)$$

or

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\sqrt{\ell(\mathbf{x})})| = \sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\sqrt{\bar{\ell}(\mathbf{x})}) + L\sqrt{\delta}| \quad (19)$$

$$= \sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\sqrt{\bar{\ell}(\mathbf{x})})| + L\sqrt{\delta} \quad (20)$$

$$\leq \delta + L\sqrt{\delta} \quad (21)$$

and therefore:

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\sqrt{\ell(\mathbf{x})})| \leq \delta + L\sqrt{\delta} \quad (22)$$

and the number of weights is at most: $\frac{6d^2R^2+3RL}{\delta}$. \square

C Folding Transformations

In this section we show how folding transformations [Szymanski and McCane, 2014] can be used to create a much deeper network with the same error, but many fewer weights than needed in Lemma 1. A folding transformation is one in which half of a space is reflected about a hyperplane, and the other half remains unchanged. Figure 1 demonstrates how a sequence of folding transformations can transform a circle in 2D to a small sector. We will use this general idea to prove the following theorem:

Theorem 1. *Let $\mathbf{x} \in \mathbb{R}^d$, and $\sigma(z) = \max(0, z)$. Let f be an L -Lipschitz function supported on $[0, R]$. Fix $L, \delta, R > 0$. There exists a function g expressible by a $O(d \log_2(d) + \log_2(d) \log_2(\frac{R}{\sqrt{\delta}}))$ layer network where the number of weights, and number of neurons, $N_w, N_n = O(d^2 + d \log_2(\frac{R}{\sqrt{\delta}}) + \frac{3RL}{\delta})$, such that:*

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\|\mathbf{x}\|)| < L\sqrt{\delta} + \delta$$

The approach taken here is a constructive one and specifies the architecture of the network needed to approximate a function. In fact, all of the weights except those in the last layer are specified. The approach is somewhat different to that used to prove Lemma 1. We build a sequence of layers to directly approximate $\|\mathbf{x}\|$ and then use Lemma 5 to approximate f . To build our layers, we need a few helper lemmas.

Lemma 2 (2D fold). *There exists a function $g : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, expressible by a ReLU network with 4 ReLU units and 2 sum units that can compute a folding transformation about a line through the origin, represented by the unit direction vector $\mathbf{l} = (l_x, l_y)^T$. The function g is of the form:*

$$g(\mathbf{x}) = \begin{cases} \mathbf{x} & \mathbf{l} \cdot \mathbf{x}^\perp > 0 \\ \begin{bmatrix} l_x^2 - l_y^2 & 2l_x l_y \\ 2l_x l_y & l_y^2 - l_x^2 \end{bmatrix} \mathbf{x} & \text{otherwise.} \end{cases}$$

Proof. The necessary ReLU network is shown in Figure 2. Only one of the nodes labeled x_- (y_-) and x_+ (y_+) are active at any one time. Therefore there are four possible cases depending on which two nodes are active. Note that x_- is active when $\mathbf{l} \cdot \mathbf{x}^\perp < 0$ and x_+ is active when $\mathbf{l} \cdot \mathbf{x}^\perp > 0$.

Case 1: x_- and y_-

$$\begin{aligned} x' &= l_y(-l_y x + l_x y) - l_x(-l_x x - l_y y) \\ y' &= -l_x(-l_y x + l_x y) - l_y(-l_x x - l_y y) \\ \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} l_x^2 - l_y^2 & 2l_x l_y \\ 2l_x l_y & l_y^2 - l_x^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned}$$

Case 2: x_+ and y_-

$$\begin{aligned} x' &= l_y(l_y x - l_x y) - l_x(-l_x x - l_y y) \\ y' &= -l_x(l_y x - l_x y) - l_y(-l_x x - l_y y) \\ \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned}$$

Case 3: x_- and y_+

$$\begin{aligned} x' &= l_y(-l_y x + l_x y) + l_x(l_x x + l_y y) \\ y' &= -l_x(-l_y x + l_x y) + l_y(l_x x + l_y y) \\ \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} l_x^2 - l_y^2 & 2l_x l_y \\ 2l_x l_y & l_y^2 - l_x^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned}$$

Case 4: x_+ and y_+

$$\begin{aligned} x' &= l_y(l_y x - l_x y) + l_x(l_x x + l_y y) \\ y' &= -l_x(l_y x - l_x y) + l_y(l_x x + l_y y) \\ \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned}$$

□

Lemma 3 (Approximate $\|\mathbf{x}\|$, $\mathbf{x} \in \mathbb{R}^2$, $\|\mathbf{x}\| < R$). *There exists a function g , expressible by a ReLU network with no more than $\log_2(R/\frac{\pi}{8})$ layers and 4 nodes per layer such that:*

$$\sup_{\mathbf{x} \in \mathbb{R}^2, \|\mathbf{x}\| \leq R} |g(\mathbf{x}) - \|\mathbf{x}\| \leq \delta$$

Proof. We simply stack layers of the type shown in Figure 2 with suitable choice of l_x, l_y at each layer. Note that the summation nodes aren't required since they can be incorporated into the summations and weights of the next ReLU layer. Call the first ReLU layer, layer 1, and set $l_{x,i} = \cos(\frac{\pi}{2^{i-1}})$, $l_{y,i} = \sin(\frac{\pi}{2^{i-1}})$. Layer 1 will fold all points to the positive y -axis half-plane. Layer 2 will fold all points to the positive (x, y) -quadrant. Layer 3 to within $\frac{\pi}{4}$ of the x -axis etc. After f such layers, $\|\mathbf{x}\|$ can be approximated with the resulting x -coordinate, with the following error:

$$\delta_1 = \|\mathbf{x}\| - \hat{x} \quad (23)$$

$$= \|\mathbf{x}\| - \|\mathbf{x}\| \cos(\frac{\pi}{2^f}) \quad (24)$$

$$\leq R(1 - \cos(\frac{\pi}{2^f})) \quad (25)$$

$$\leq R(2 \sin^2(\frac{\pi}{2^{f+1}})) \quad (26)$$

$$\leq R(2 \sin(\frac{\pi}{2^{f+1}})) \quad (27)$$

$$\leq R(\frac{\pi}{2^f}) \quad (28)$$

$$2^f \leq R \frac{\pi}{\delta_1} \quad (29)$$

$$f \leq \log_2(R \frac{\pi}{\delta_1}) \quad (30)$$

□

Lemma 4 (Approximate $\|\mathbf{x}\|, x \in \mathbb{R}^d$). *There exists a function g , expressible by a ReLU network with:*

$$\begin{aligned} N_l &\leq \log_2(d) \\ &\log_2 \left(\frac{R\pi}{\delta} \left[(2^{\lfloor \frac{d+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{d}{2} \rfloor} - 1) \right] \right) \\ N_n &\leq 4(d-1) \\ &\log_2 \left(\frac{R\pi}{\delta} \left[(2^{\lfloor \frac{d+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{d}{2} \rfloor} - 1) \right] \right) \\ N_w &\leq 8(d-1) \\ &\log_2 \left(\frac{R\pi}{\delta} \left[(2^{\lfloor \frac{d+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{d}{2} \rfloor} - 1) \right] \right) \end{aligned}$$

such that:

$$\sup_{\mathbf{x} \in \mathbb{R}^d, \|\mathbf{x}\| \leq R} |g(\mathbf{x}) - \|\mathbf{x}\|| \leq \delta$$

Proof. We note that a fold in 2D plane in \mathbb{R}^d will leave all coordinates perpendicular to the plane unchanged. We can therefore apply the approximation of Lemma 3 to pairs of input coordinates to produce $d/2$ new coordinates. Then apply the same reduction

to produce $d/4$ coordinates and continue on this way until there is only one coordinate left. In effect, we are calculating the norm via the following scheme:

$$\sqrt{x_1^2 + x_2^2 + x_3^2 + \dots + x_d^2} = \sqrt{\dots \sqrt{\sqrt{x_1^2 + x_2^2}^2 + \sqrt{x_3^2 + x_4^2}^2} \dots \sqrt{\sqrt{\dots}^2 + \sqrt{x_{n-1}^2 + x_{n-2}^2}^2}} \quad (31)$$

Let g_i represent the function mapping of fold sequence i . That is, $g_{1,x_{1:2}}$ denotes the first fold sequence that folds the first two coordinates x_1, x_2 ; $g_{1,x_{3:4}}$ folds the second two coordinates etc. And $g_{2,x_{1:4}}$ denotes the second fold sequence that folds the output of the first two folds in layer one. See Figure 3 for a schematic of the network. More formally, we have (with some abuse of notation):

$$g_{1,j:j+1} = g_1(x_j, x_{j+1})$$

$$g_{i>1,j:j+2^i-1} = g_i(g_{i-1,x_{j:j+2^{i-1}-1}}, g_{i=1,x_{j+2^{i-1}-1:j+2^i-1}})$$

Each fold layer requires $\log_2(R_{\delta_1}^{\pi})$ network layers of 4 neurons each and results in an error no greater than δ_1 . We have the following situation after the first fold layer:

$$\sqrt{x_1^2 + x_2^2} - \delta_1 \leq g_{1,x_{1:2}} \leq \sqrt{x_1^2 + x_2^2} + \delta_1$$

$$\sqrt{x_3^2 + x_4^2} - \delta_1 \leq g_{1,x_{3:4}} \leq \sqrt{x_3^2 + x_4^2} + \delta_1$$

$$\dots$$

We proceed via induction and bound the error produced at each subsequent layer:

$$\begin{aligned} & \sqrt{\sum_{j=n}^{n+2^i-1} x_j^2} - \left[(2^{\lfloor \frac{i+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{i}{2} \rfloor} - 1) \right] \delta_1 \\ & \leq g_{i,x_{n:n+2^i-1}} \\ & \leq \sqrt{\sum_{j=n}^{n+2^i-1} x_j^2} + \left[(2^{\lfloor \frac{i+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{i}{2} \rfloor} - 1) \right] \delta_1, \end{aligned} \quad (32)$$

where n is the appropriate coordinate index, $2 \leq i \leq d$, δ_1 is the error from a single fold. Appropriate coordinate index in this context means $n \in \{1, 5, 9, \dots\}$ for $i = 2$, $n \in \{1, 9, 17, \dots\}$ for $i = 3$, etc.

At the second fold layer (the base case, $i = 2$), we would like to bound the error on the result in terms of the target computation ($\sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2}$). We use the first 4 coordinates without loss of generality. The second fold layer gives us:

$$\sqrt{g_{1,x_{1:2}}^2 + g_{1,x_{3:4}}^2} - \delta_1 \leq g_{2,x_{1:4}} \leq \sqrt{g_{1,x_{1:2}}^2 + g_{1,x_{3:4}}^2} + \delta_1$$

First, consider the right hand side:

$$\begin{aligned}
& g_{2,x_{1:4}} \\
& \leq \sqrt{g_{1,x_{1:2}}^2 + g_{1,x_{3:4}}^2} + \delta_1 \\
& \leq \sqrt{(\sqrt{x_1^2 + x_2^2} + \delta_1)^2 + (\sqrt{x_3^2 + x_4^2} + \delta_1)^2} + \delta_1 \\
& \leq \sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2 + 2\delta_1(\sqrt{x_1^2 + x_2^2} + \sqrt{x_3^2 + x_4^2}) + 2\delta_1^2} + \delta_1 \\
& \leq \sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2 + 2\sqrt{2}\delta_1(\sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2}) + 2\delta_1^2} + \delta_1 \\
& \leq \sqrt{(\sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2} + \sqrt{2}\delta_1)^2} + \delta_1 \\
& \leq \sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2} + (1 + \sqrt{2})\delta_1
\end{aligned}$$

The left hand side:

$$\begin{aligned}
& g_{2,x_{1:4}} \\
& \geq \sqrt{g_{1,x_{1:2}}^2 + g_{1,x_{3:4}}^2} - \delta_1 \\
& \geq \sqrt{(\sqrt{x_1^2 + x_2^2} - \delta_1)^2 + (\sqrt{x_3^2 + x_4^2} - \delta_1)^2} - \delta_1 \\
& \geq \sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2 - 2\delta_1(\sqrt{x_1^2 + x_2^2} + \sqrt{x_3^2 + x_4^2}) + 2\delta_1^2} - \delta_1 \\
& \geq \sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2 - 2\sqrt{2}\delta_1(\sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2}) + 2\delta_1^2} - \delta_1 \\
& \geq \sqrt{(\sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2} - \sqrt{2}\delta_1)^2} - \delta_1 \\
& \geq \sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2} - (1 + \sqrt{2})\delta_1
\end{aligned}$$

These clearly satisfy Equation 32.

For the induction step we want to show that Equation 32 is true for $i + 1$, given it is true for i . We start by applying Lemma 2:

$$\begin{aligned}
& \sqrt{g_{i,x_{n:n+2^i-1}}^2 + g_{i,x_{n+2^i:n+2^{i+1}-1}}^2} - \delta_1 \\
& \leq g_{i+1,x_{n:n+2^{i+1}-1}} \\
& \leq \sqrt{g_{i,x_{n:n+2^i-1}}^2 + g_{i,x_{n+2^i:n+2^{i+1}-1}}^2} + \delta_1
\end{aligned} \tag{33}$$

Start with the right hand side and for brevity just use g_{i+1} , and let $\delta_i = \left[(2^{\lfloor \frac{i+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{i}{2} \rfloor} - 1) \right] \delta_1$,

$$a = \sum_{j=n}^{n+2^i-1} x_j^2, b = \sum_{j=n+2^i}^{n+2^{i+1}-1} x_j^2;$$

$$\begin{aligned} g_{i+1} &\leq \sqrt{(\sqrt{a} + \delta_i)^2 + (\sqrt{b} + \delta_i)^2} + \delta_1 \\ &\leq \sqrt{a + b + 2\delta_i(a + b) + 2\delta_i^2} + \delta_1 \\ &\leq \sqrt{(\sqrt{a + b} + \sqrt{2}\delta_i)^2} + \delta_1 \\ &\leq \sqrt{a + b} + \sqrt{2}\delta_i + \delta_1 \\ &\leq \sqrt{a + b} + \sqrt{2} \left[(2^{\lfloor \frac{i+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{i}{2} \rfloor} - 1) \right] \delta_1 + \delta_1 \\ &\leq \sqrt{a + b} + \left[\sqrt{2}((2^{\lfloor \frac{i+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{i}{2} \rfloor} - 1)) + 1 \right] \delta_1 \\ &\leq \sqrt{a + b} + \left[(2^{\lfloor \frac{i+2}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{i+1}{2} \rfloor} - 1) \right] \delta_1 \end{aligned}$$

As required. A similar argument can be constructed for the left hand side. Each folding sequence requires $\log_2(R \frac{\pi}{\delta_1})$ layers and results in an error no greater than δ_1 . We need $\log_2 d$ such folding sequences giving the number of layers as:

$$N_l = \log_2 d \log_2(R \frac{\pi}{\delta_1}) \quad (34)$$

The first fold sequence requires $2d$ neurons per layer and $4d$ weights per layer. The second fold sequence requires d neurons per layer and $2d$ weights per layer and so on. The total number of neurons then is:

$$\begin{aligned} N_n &= \sum_{i=1}^{\log_2 d} \frac{2d}{2^{i-1}} \log_2(R \frac{\pi}{\delta_1}) \\ &= 4(d-1) \log_2(R \frac{\pi}{\delta_1}) \end{aligned}$$

The total number of weights, then is:

$$N_w = 8(d-1) \log_2(R \frac{\pi}{\delta_1})$$

In terms of the overall error of the approximation (δ), we have:

$$\begin{aligned} N_l &= \log_2 d \log_2 \left(R \left[(2^{\lfloor \frac{d+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{d}{2} \rfloor} - 1) \right] \frac{\pi}{\delta} \right) \\ &= O \left(d \log_2 d + d \log_2 \left(R \frac{\pi}{\delta} \right) \right) \\ N_n &= 4(d-1) \log_2 \left(R \left[(2^{\lfloor \frac{d+1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{d}{2} \rfloor} - 1) \right] \frac{\pi}{\delta} \right) \\ &= O \left(d^2 + d \log_2 \left(R \frac{\pi}{\delta} \right) \right) \\ N_w &= O \left(d^2 + d \log_2 \left(R \frac{\pi}{\delta} \right) \right) \end{aligned}$$

□

Now we are ready to prove the main theorem.

Theorem 1. *Let $\mathbf{x} \in \mathbb{R}^d$, and $\sigma(z) = \max(0, z)$. Let f be an L -Lipschitz function supported on $[0, R]$. Fix $L, \delta, R > 0$. There exists a function g expressible by a $O(d \log_2(d) + \log_2(d) \log_2(\frac{R}{\sqrt{\delta}}))$ layer network where the number of weights, and number of neurons, $N_w, N_n = O(d^2 + d \log_2(\frac{R}{\sqrt{\delta}}) + \frac{3RL}{\delta})$, such that:*

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - f(\|\mathbf{x}\|)| < L\sqrt{\delta} + \delta$$

Proof. From Lemma 4 we can approximate $\|\mathbf{x}\|$ to within $\sqrt{\delta}$. Therefore:

$$\begin{aligned} f(\|\mathbf{x}\| + \sqrt{\delta}) - \delta &\leq g(\|\mathbf{x}\| + \sqrt{\delta}) \leq f(\|\mathbf{x}\| + \sqrt{\delta}) + \delta, \text{ from Lemma 5} \\ f(\|\mathbf{x}\|) + L\sqrt{\delta} - \delta &\leq g(\|\mathbf{x}\| + \sqrt{\delta}) \leq f(\|\mathbf{x}\|) + L\sqrt{\delta} + \delta \end{aligned}$$

and

$$\begin{aligned} f(\|\mathbf{x}\| - \sqrt{\delta}) - \delta &\leq g(\|\mathbf{x}\| - \sqrt{\delta}) \leq f(\|\mathbf{x}\| - \sqrt{\delta}) + \delta \\ f(\|\mathbf{x}\|) - L\sqrt{\delta} - \delta &\leq g(\|\mathbf{x}\| - \sqrt{\delta}) \leq f(\|\mathbf{x}\|) - L\sqrt{\delta} + \delta \end{aligned}$$

therefore:

$$f(\|\mathbf{x}\|) - L\sqrt{\delta} - \delta \leq g(\|\mathbf{x}\| + \sqrt{\delta}) \leq f(\|\mathbf{x}\|) + L\sqrt{\delta} + \delta$$

The number of weights and neurons required by Lemma 5 is $\frac{3RL}{\delta}$. The number of weights and neurons required to estimate $\|\mathbf{x}\|$ is given by Lemma 4 (substituting $\sqrt{\delta}$ for δ). Stack the network from Lemma 5 onto the end of the network from Lemma 4, thus requiring a total number of neurons no more than:

$$\begin{aligned} N_n &\leq \left[4(d-1) \log_2 \left(\frac{R\pi}{\sqrt{\delta}} \left[(2^{\lfloor \frac{d-1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{d}{2} \rfloor} - 1) \right] \right) \right] + \frac{3RL}{\delta} \\ N_n &= O(d^2 + d \log_2(\frac{R}{\sqrt{\delta}}) + \frac{3RL}{\delta}) \end{aligned}$$

and a total number of weights no more than:

$$\begin{aligned} N_w &\leq \left[8(d-1) \log_2 \left(\frac{R\pi}{\sqrt{\delta}} \left[(2^{\lfloor \frac{d-1}{2} \rfloor} - 1) + \sqrt{2}(2^{\lfloor \frac{d}{2} \rfloor} - 1) \right] \right) \right] + \frac{3RL}{\delta} \\ N_w &= O(d^2 + d \log_2(\frac{R}{\sqrt{\delta}}) + \frac{3RL}{\delta}) \end{aligned}$$

□